ENABLING MACHINE LEARNING AS A SERVICE (MLAAS) WITH GPU ACCELERATION USING VMWARE VREALIZE AUTOMATION

Technical White Paper

July 2018

vmware®

Index

1. MACHINE LEARNING AS A SERVICE (MLAAS) IN PRIVATE CLOUD

Machine learning (ML) is a capability that enables computers to learn without being explicitly programmed (Arthur Samuel, 1959). While the concept of ML has existed for decades, dramatic increases in computing capability and data volume have recently accelerated its development. A typical ML workflow includes: data-cleaning, model-selection, feature-engineering, model-training, and inference. Building and maintaining production ML environments is often complex since each ML process may require customization of the hardware and software. Much of this complexity can be eliminated by automating the deployment of hardware resources, configuring them with the appropriate operating system and application stack, and providing them to data scientists. This simplification process is referred to as Machine Learning as a Service (MLaaS).

Adoption of virtualization for High Performance Computing (HPC) and big data workloads has grown rapidly due to a number of benefits, including simultaneous support of mixed software environments, cluster resource-sharing, research environment reproducibility, multi-tenant data security, fault-isolation and resiliency, dynamic load-balancing, and performance. As a result of this trend, and the convergence of HPC and big data, ML is increasingly run in cloud environments. Due to the large volumes of data involved, as well as data compliance and security issues and the large amounts of required computation time, organizations are increasingly focused on using private cloud for hosting their ML workloads.

This white paper presents step-by-step guidance to illustrate how administrators can enable MLaaS in a private cloud using VMware vRealize Automation to offer GPU-accelerated ML services for data scientists. The workflow shown here focuses on TensorFlow™, but is easily adapted to create other ML-based templates that can address the specific needs of an organization's data scientists.

2. VMWARE APPROACHES FOR MLAAS

There are two private-cloud approaches for providing Infrastructure as a Service (IaaS) in a VMware environment: VMware vRealize Automation and VMware Integrated OpenStack. vRealize Automation is a cloud-management platform that enables IT automation through the creation and management of customized infrastructure, applications, and services deployed rapidly across multi-vendor, multi-cloud environments. VMware Integrated OpenStack is a VMware-supported OpenStack distribution that makes it easy for IT to run an enterprise-grade OpenStack cloud on top of VMware virtualization technologies, boosting productivity by providing simple, standard and vendor-neutral OpenStack API access to VMware infrastructure.

This paper is focused on creating MLaaS using VMware vRealize Automation.

3. GENERAL-PURPOSE GPU (GPGPU)

TThe use of GPUs as parallel computing accelerators, or GPGPUs, is a significant trend within HPC. Since 2012, NVIDIA GPUs have been commonly used for ML and Deep Learning (DL) due to the massive parallelism of GPGPUs, which is naturally beneficial for computationally expensive workloads like ML/DL. The CUDA (Compute Unified

Device Architecture) toolkit is an NVIDIA development environment for GPGPU computing. CuDNN is a deep neural network library based on CUDA that allows developers to easily implement deep neural networks with optimized performance rather than having to build them from lower-level CUDA calls.

NVIDIA GPUs can be enabled in two ways as compute accelerators in the VMware vSphere environment: VMware DirectPath I/O and NVIDIA Grid virtual GPU. Both of these methods are described in turn below.

3.1 VMware DirectPath I/O

In vSphere, PCI devices can be configured in DirectPath I/O (passthrough) mode, which allows a guest OS to directly access the device, essentially bypassing the hypervisor. Because of the shortened access path, performance of applications accessing GPUs in this way can be very close to that of bare-metal systems. With DirectPath I/O, we can configure one or multiple NVIDIA GPU devices into a single VM.

The steps to configure Direct Path I/O are:

1. Enable passthrough on a host
2. Configure a PCI device in a VM

For some devices with large PCI configuration spaces (base address registers, commonly called BARs), special additional configuration steps are needed. For devices commonly used for ML, such as the NVIDIA K80, P100, and V100, or if you are having difficulty successfully configuring a device in passthrough mode, see "How to Enable Compute Accelerators on vSphere 6.5 for Machine Learning and Other HPC Workloads."

3.2 NVIDIA Virtual GPU

The second method to enable GPGPU acceleration uses the NVIDIA GRID vGPU solution, also called NVIDIA Quadro Virtual Data Center Workstation (Quadro vDWS). NVIDIA GRID enables sharing of an NVIDIA Tesla GPU card across multiple VMs by creating multiple, logical vGPU devices, each of which can be assigned to a VM. vGPUs are created using profiles which specify how much GPU memory has been assigned to a vGPU. Currently, all vGPUs created on the same physical GPU must use the same profile. Note that while GPU memory is partitioned to create vGPUs, each vGPU is given time-sliced access to the entirety of a GPU's compute resources. Time-slicing is managed using one of several available scheduling algorithms, each appropriate for different shared GPU use cases.

Use of NVIDIA GRID vGPU on vSphere requires installation of an ESXi driver and several additional configuration steps which are described in detail in later sections of this white paper.

With proper sizing enforcements, vGPUs can be enabled as a service for cloud consumption using vRealize Automation. This is also explained in detail later in this document.

Table 1 summarizes the configuration requirements and IaaS support for cloud consumption within each accelerator option.

# of NVIDIA physical GPUs per VM	METHOD	REQUIREMENTS	VMWARE IAAS
≥ 1	Direct Path I/O	If the GPU device has large configuration space, like NVIDIA K40, K80, P100, special configurations are needed. See this VMware OCTO Blog and KB 1018392.	VIO
<1 (shared)	NVIDIA vGPU	Applicable to vDWS-supported NVIDIA cards; see NVIDIA documentation.	vRA (with sizing enforcements) or VIO

Table 1. Scenarios for using NVIDIA GPUs for compute usage (with CUDA)

4. MLAAS USING VREALIZE AUTOMATION AND vGPU

In this section, the steps to define and enable a TensorFlow service for self-provisioned, end-user consumption using the NVIDIA GRID vGPU solution with vRealize Automation are are outlined.

4.1 Architecture

Figure 1 illustrates the key components in this example. The hardware includes an NVIDIA P100 card and a compatible host. The software stack includes vSphere, vRealize Automation, NVIDIA Grid drivers to enable NVIDIA vGPU, and a guest OS distribution. Using this base architecture, administrators can install Docker and nvidia-docker to pull images that contain CUDA/cuDNN and a variety of ML-related toolkits/libraries to meet end-user requirements.

In this example, four vGPUs have been created on the P100 GPU. Since the P100 has 16GB of GPU memory, each vGPU has been allocated 4GB. In NVIDIA terminology, these vGPUs are using GRID_P100-4Q profiles, where "4Q" refers to the vGPU's memory size. For more information regarding the profiles, please refer to NVIDIA virtual GPU user guide.

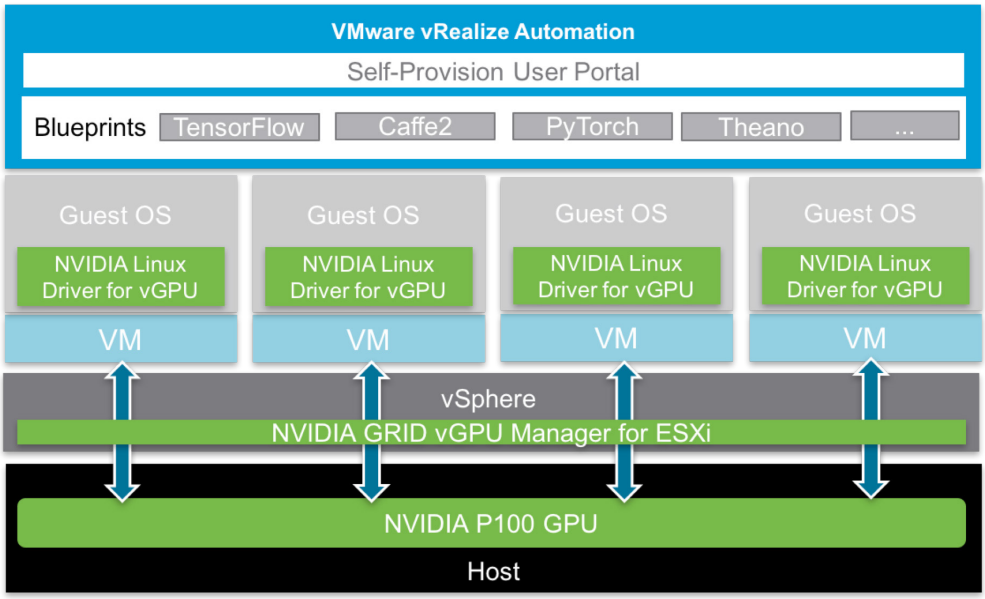

Figure 1. Illustration of the components in this example

4.2 NVIDIA vGPU Configuration on vSphere ESXi

As a prerequisite, the NVIDIA GRID Virtual GPU Manager for ESXi driver must be installed on ESXi to enable use of vGPUs. In addition, the physical GPU needs to be configured in ESXi as a vGPU device, as opposed to the default shared graphics (vSGA) device. The configuration steps are:

1) Enable vGPU mode on the ESXi host and verify

```
# esxcli graphics host set --default-type SharedPassthru
# esxcli graphics host get
   Default Graphics Type: SharedPassthru
   Shared Passthru Assignment Policy: Performance
```

2) Install the NVIDIA GRID driver VIB on ESXi

```
# esxcli software vib install --maintenance-mode --no-sig-check -v "/vib-dir/NVIDIA-NVIDIA-VMware-384.73-1OEM.650.0.0.4598673.x86_64.vib"
```

vib-dir is the directory where installation VIB should be located, downloadable directly from NVIDIA

3) In the ESXi shell, verify that the driver is working properly by using the nvidia-smi utility, which confirms that the card is configured in vGPU mode

```
#  nvidia-smi -q | grep -i virtualization
   GPU Virtualization Mode
      Virtualization mode         : Host VGPU
```

4) In the ESX shell, list the vGPU profiles supported by the GPU

```
# nvidia-smi vgpu --supported
GPU 00000000:03:00.0
    GRID P100-1Q
    GRID P100-2Q
    GRID P100-4Q
    GRID P100-8Q
    GRID P100-16Q
    GRID P100-1A
    GRID P100-2A
    GRID P100-4A
    GRID P100-8A
    GRID P100-16A
    GRID P100-1B
```

5) Disable ECC mode, which NVIDIA does not currently support for vGPU

```
# nvidia-smi -e 0
Disabled ECC support for GPU 00000000:03:00.0.
All done.
```

4.3 NVIDIA vGPU Configuration on vSphere ESXi

4.3.1 Overview

To create a vRealize Automation blueprint that end users can invoke to self-provision a TensorFlow service, several steps must be followed by the vSphere administrator and/or the vRealize Automation blueprint architect. The workflow for preparing and publishing a TensorFlow service is illustrated in Figure 2. Starting with a CentOS virtual machine, there are five steps needed to create and publish a TensorFlow service for end users:

1. Add an NVIDIA GRID vGPU to the VM
2. Customize the guest OS environment
3. Convert the VM to a template and create a customization specification
4. Design the blueprint
5. Publish the blueprint

The first three steps are performed within vSphere and the last two steps are completed within vRealize Automation. The following sections will explain each of the steps shown in the figure in detail.

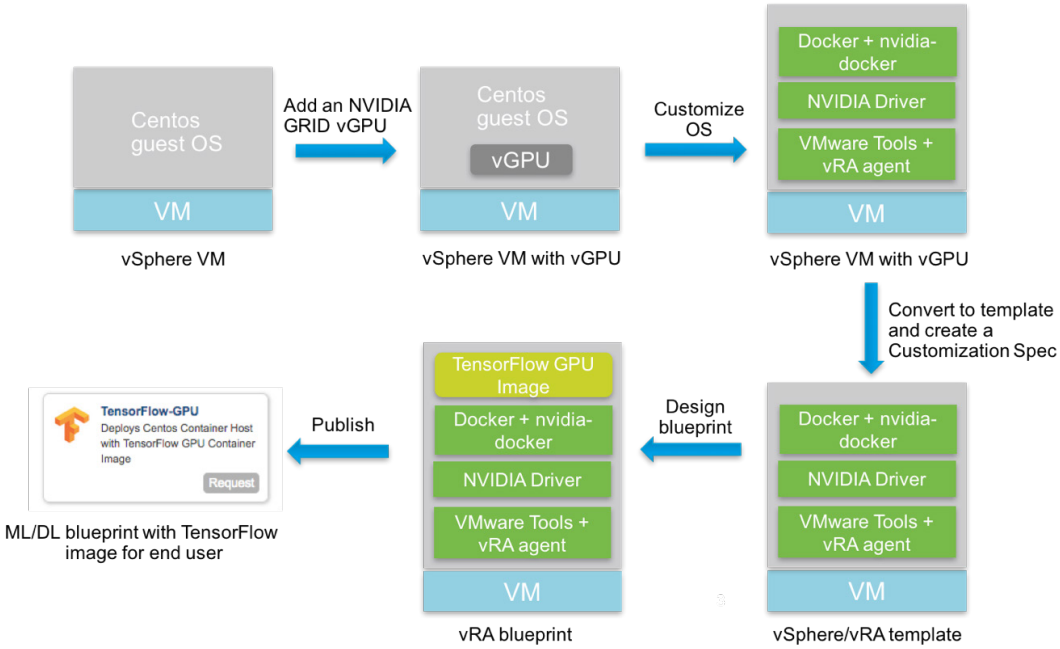

Figure 2. Workflow for preparing and publishing a vRealize Automation TensorFlow service

4.3.2 Add an NVIDIA GRID vGPU

Begin with a VM with the CentOS OS 7.2 installed. To create the TensorFlow service, first add an NVIDIA GRID vGPU into the VM using vCenter as shown in Figure 3. Once the vGPU has been specified, an appropriate vGPU profile can be chosen from the GPU Profile dropdown menu.

NOTE: Adding a PCI device, including a vGPU, requires that all VM memory be reserved. In addition, some VM operations are unavailable when passthrough devices are present (e.g., vMotion and VM snapshots).

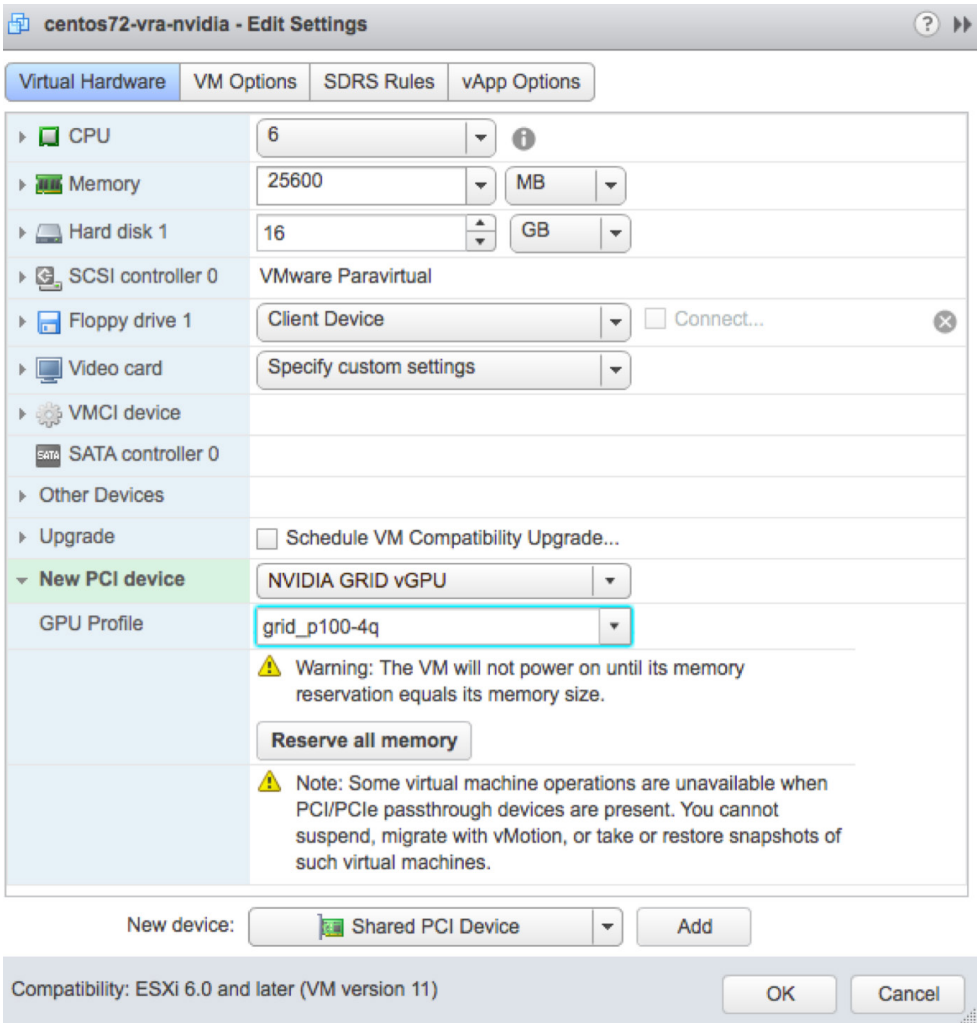

Figure 3. Configuring a vGPU in a vSphere VM

4.3.3 Customize the OS Environment

Once the VM has been configured with an NVIDIA GRID vGPU device, customize the OS environment by pre-installing the necessary tools and libraries into the VM so it can later be converted to a vRealize Automation template. Both VMware tools and the vRealize Automation agent are prerequisites. To install or upgrade VMware Tools on a Linux system, please see VMware KB article 1018414 or KB article 1018392. In addition, the NVIDIA guest OS driver, Docker, and nvidia-docker can either be pre-installed in the template or post-installed later as a vRealize Automation **software component**. Since these are common tools for preparing a wide variety of data-science environments, pre-installing them into the template is recommended.

The steps for installing the above components in CentOS are shown below, having first logged into a bash shell as root.

Update CentOS to the newest versions of packages, using skip-broken to skip packages that have dependency problems or introduce problems to the installed packages.

```
# yum update --skip-broken -y
```

Install the vRA guest agent to prepare for software provisioning, specifying the actual fully-qualified domain name (FQDN) from which to retrieve the agent, and specifying appropriate IP addresses or hostnames for the deployment. When running the vra_template script, the cloud provider type should be set to vsphere.

```
# wget --no-check-certificate --directory-prefix=/tmp  \
  https://vRealize_VA_hostname_fqdn/software/download/prepare_vra_template.sh
# chmod +x prepare_vra_template.sh
# ./prepare_vra_template.sh
```

Install the NVIDIA Linux driver for vGPU, following prompts to finish the installation.

```
# sh /tmp/NVIDIA-Linux-x86_64-384.73-grid.run
```

Verify the NVIDIA driver is working properly using the nvidia-smi utility, which should output the driver version, profile name, and some monitoring statistics.

```
# nvidia-smi
+-----------------------------------------------------------------------------------------------+
| NVIDIA-SMI 384.73                 Driver Version: 384.73                                      |
|-------------------------------+----------------------+-----------------------------------+
| GPU  Name            Persistence-M|  Bus-Id         Disp.A | Volatile Uncorr. ECC          |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.          |
|===============================+======================+===========|
|   0  GRID P100-4Q        On   | 00000000:02:02.0 Off |                N/A                |
| N/A      P0    N/A /  N/A |    272MiB /  4095MiB |      0%      Default                   |
+-------------------------------+----------------------+-----------------------------------+

+-----------------------------------------------------------------------------------------------+
| Processes:                                                                       GPU Memory   |
|  GPU       PID   Type   Process name                                             Usage        |
|===============================================================================|
|  No running processes found                                                                  |
+-----------------------------------------------------------------------------------------------+
```

Follow the NVIDIA GRID License guide to license NVIDIA GRID vGPU usage on Linux.

Install Docker Community Edition (CE) and enable its systemd service. For more information about Docker CE, see the user guide.

```
# yum install -y yum-utils device-mapper-persistent-data lvm2
# yum-config-manager \
  --add-repo https://download.docker.com/linux/centos/docker-ce.repo
# yum install -y docker-ce-17.09.1.ce
# systemctl enable docker
```

Install nvidia-docker. Note that there is a version dependency between nvidia-docker and Docker and that this installation step may require installation of a different version of Docker CE. Read further details about nvidia-docker installation.

```
# curl -s -L \
  https://nvidia.github.io/nvidia-docker/centos7/x86_64/nvidia-docker.repo | \
  sudo tee /etc/yum.repos.d/nvidia-docker.repo
# yum install -y nvidia-docker2
# pkill -SIGHUP dockerd
```

Now power-off the VM in preparation for converting it to a template.

4.3.4 Design the vRealize Automation Blueprint

With the driver and all tools installed and the vGPU configured, power off the VM and convert it to a template, which can then be used as the basis for a blueprint by vRealize Automation. Once the template is created, make a standard **customization specification** for the vRealize Automation IaaS architect to use when creating clone blueprints for TensorFlow VMs.

The procedure for creating a template from a VM is as follows:

1. Log into your vSphere web client as administrator
2. Right click the VM object and select Template -> Convert to Template
3. On the home page, click Customization Specification Manager and create a new specification based on the template

4.3.5 Design the vRealize Automation Blueprint

There are two steps needed to design the vRealize Automation blueprint. First, a **software component** should be created that defines the software lifecycle for additional software to be dynamically installed and configured within a blueprint at VM creation time. Second, the blueprint itself must be built using the design functionality of vRealize Automation.

1) Define a Software Component

In vRealize Automation, the blueprint architect can build software components to define how software will be installed, configured, started, updated, and uninstalled within a blueprint's VM. Once a component is defined, drag-and-drop it onto appropriate container types on the design canvas, as will be covered in a later step.

In this section, building a simple component that can install TensorFlow is demonstrated:

1. Click the **Design** tab
2. Click on **Software Components**
3. Click **New**

In the general section shown in **Figure 4**, add a **name** and **description** for the software component. The container type should be specified as machine. Click **next** to continue.

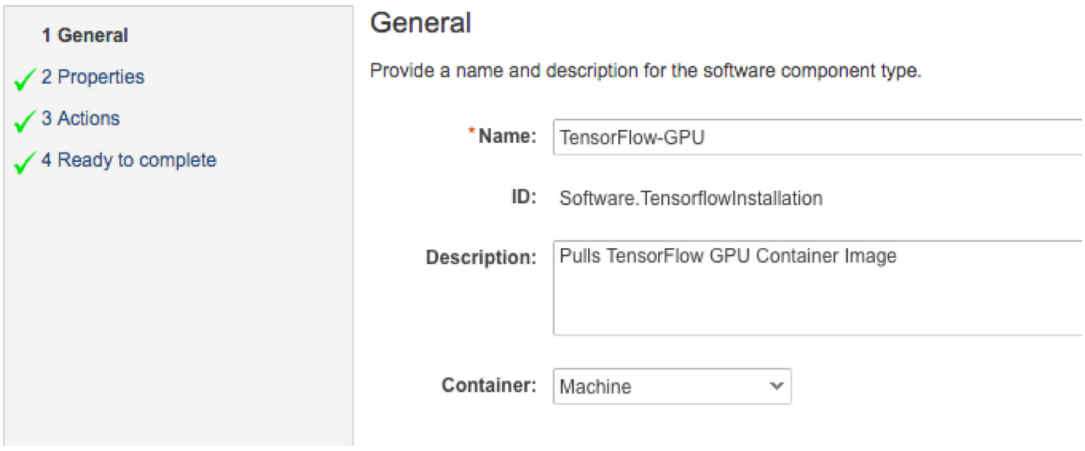

Figure 4. Creating a vRealize Automation Software Component (General tab)

In the actions section shown in Figure 5, define the **install lifecycle stage** action, provide a command script that will pull a TensorFlow GPU container image, and also modify the /etc/motd file to display instructions to the end user when logging into the TensorFlow VM. In this example, we chose to use a vRealize Automation Software Component to install the container image. vRealize Automation has a well-integrated Container Management capability that could have been used instead. The example script can be customized to perform more complex configuration and installation steps as desired. The following example uses a bash script, so this should be selected as the **script type**.

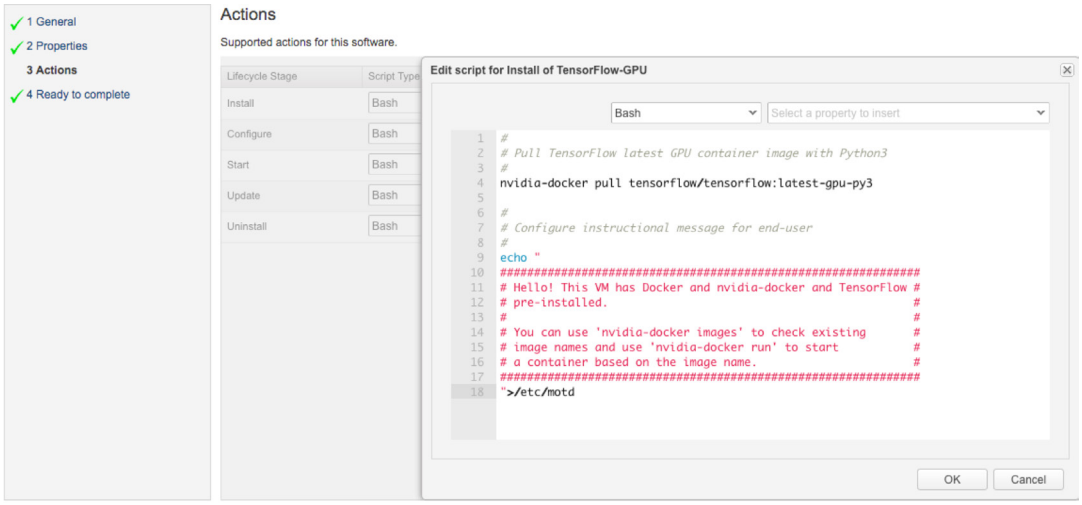

Figure 5. Creating a vRealize Automation Software Component (Action tab)

```
#
# Pull TensorFlow latest GPU container image with Python3
#
nvidia-docker pull tensorflow/tensorflow:latest-gpu-py3

#
# Configure instructional message for end-user
#
echo "
############################################################
# Hello! This VM has Docker and nvidia-docker and TensorFlow #
# pre-installed.                                           #
#                                                          #
# You can use 'nvidia-docker images' to check existing     #
# image names and use 'nvidia-docker run' to start         #
# a container based on the image name.                     #
############################################################
">/etc/motd
```

1. Click **Next**
2. Click **Finish**

Tensorflow-GPU should now appear in the software components section, but it must be published before it can be used in the creation of a blueprint. To do this:

1. In Software Components select **Tensorflow-GPU**
2. Click **Publish**

2) Design Blueprint
Blueprints are used to define how services are deployed using vRealize Automation. A basic blueprint might create only a single VM while a more complex blueprint might create multiple VMs, software components, networks, security policies, and the like. The blueprint shown below is relatively simple, containing only a CentOS VM, the Tensorflow software component, and an existing virtual network:

1. Click the **Design** tab
2. Click **Blueprints**
3. Click **New**

From the **Design** tab, select **blueprints** and then **new**. In the **new blueprint** section insert the **name** and **description** (optional). An ID will be created automatically based on the name. This is shown in Figure 6.

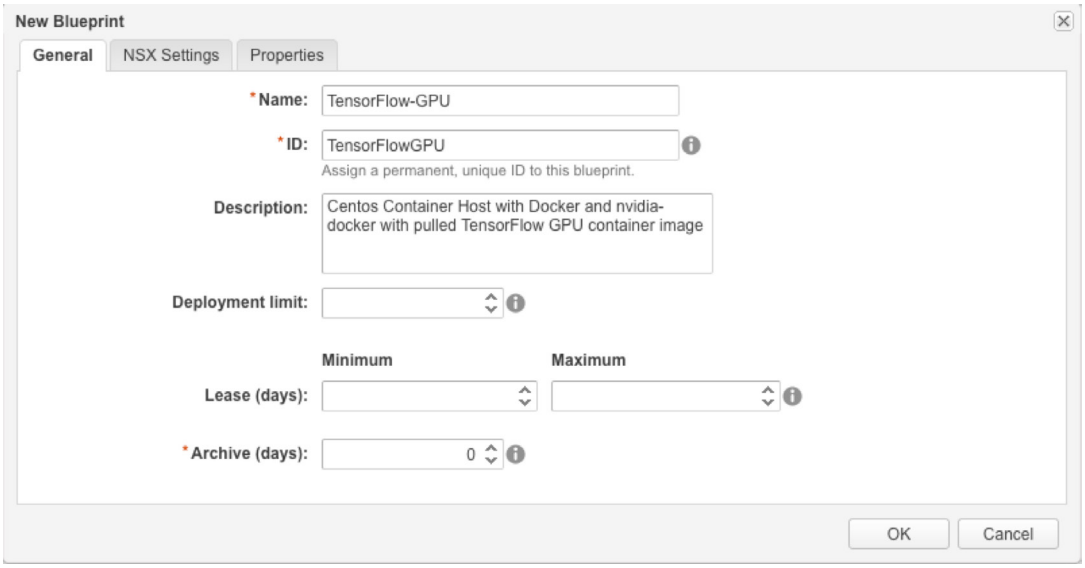

Figure 6. Creating a new vRealize Automation blueprint

Next, add a vSphere machine to the TensorFlow blueprint by selecting **machine types** from the **categories** section and clicking and dragging **vSphere (vCenter) Machine** to the design canvas as shown in Figure 7.

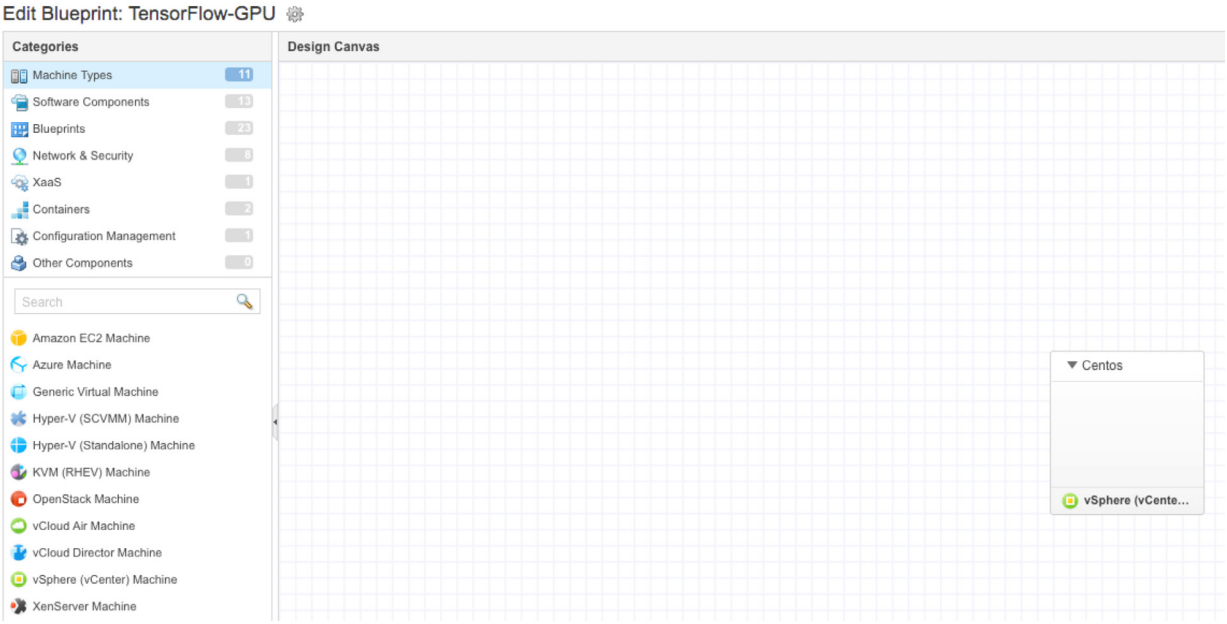

Figure 7. Adding a machine to the blueprint

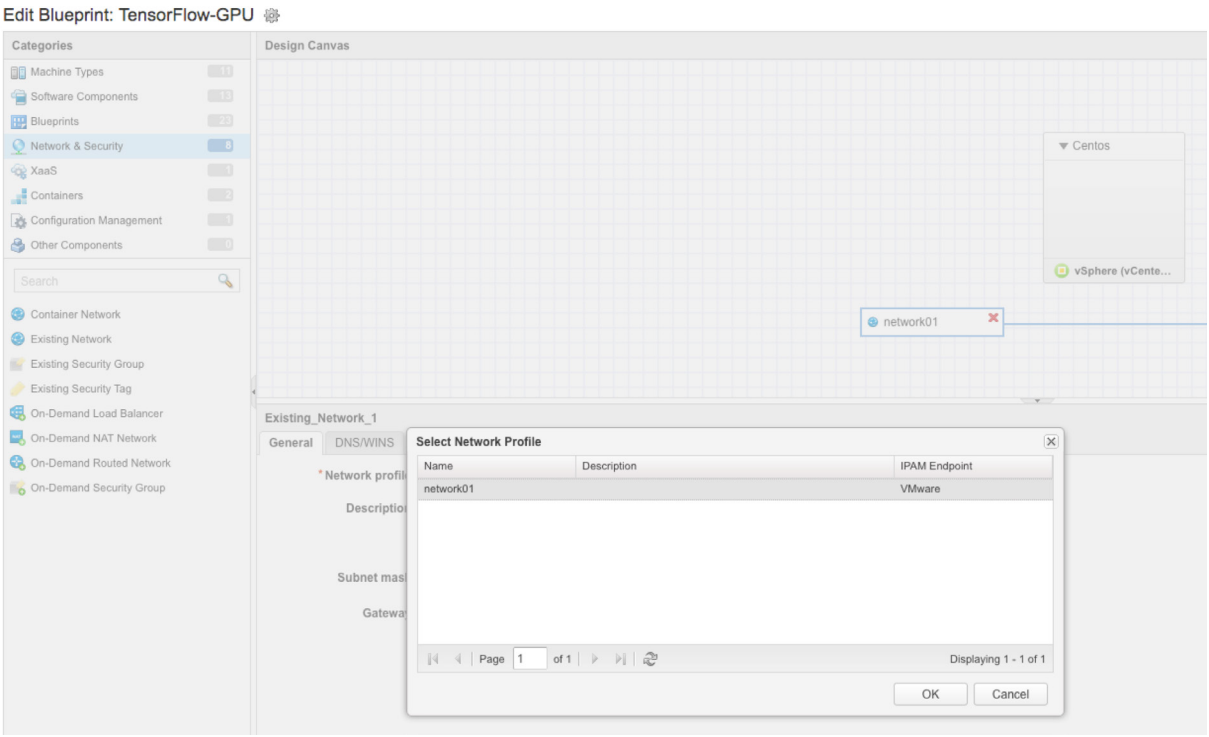

Figure 8. Blueprint network configuration

Next, select the vSphere machine on the canvas and configure additional properties. In Figure 9 we show the **General** tab where we have provided a prefix to be used on every VM name created by the blueprint and have allowed users to self-provision up to four TensorFlow VMs.

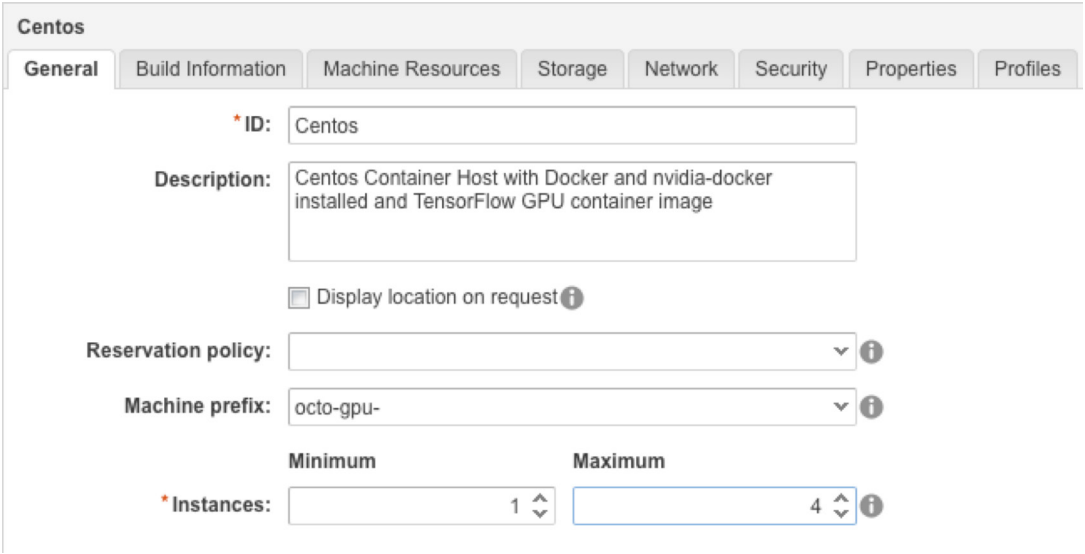

Figure 9. Configuring additional parameters (**General** tab)

Since the VM template is configured with a vGPU, which is a passthrough PCI device, some vSphere features are unavailable, including snapshots as well as linked clones, which depend on snapshots. Therefore, this blueprint will need to be configured to use a full clone of the centos72-vra-nvidia VM template created earlier. To do this, move to the **Build Information** tab and fill in fields as shown in Figure 10.

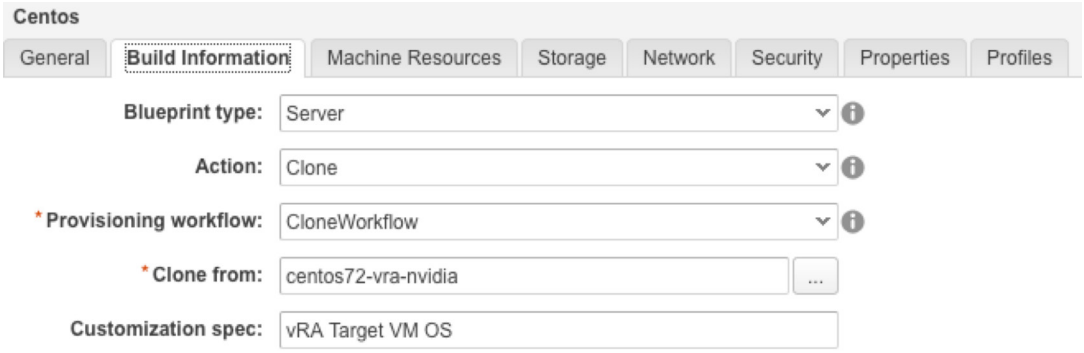

Figure 10. Configuring additional parameters (Build Information tab)

Now proceed to the **Machine Resources** tab and provide information about the virtual CPU, memory and storage to be allocated for Virtual Machines created by this blueprint. The values used in the example are shown in Figure 11.

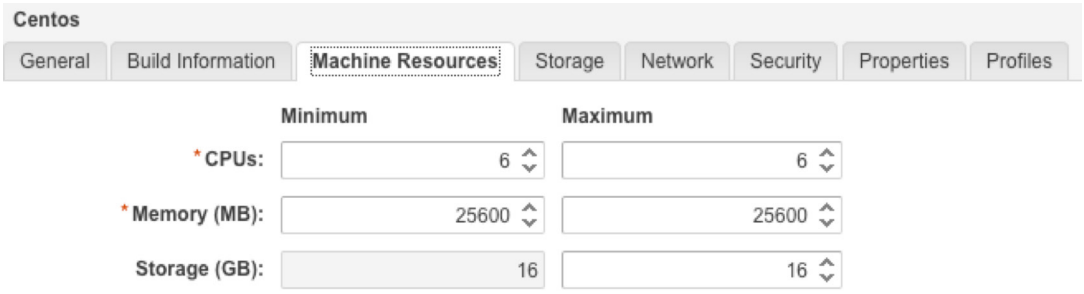

Figure 11. Configuring additional parameters (Machine Resources tab)

Under the **Network** tab, specify which existing virtual network the new VM should be attached to when it is created. To do this, click **new**, select the appropriate network from the dropdown, select **DHCP** or **static** and then click **OK**. See Figure 12.

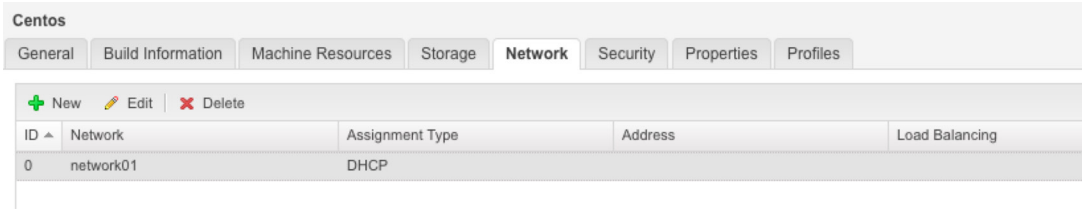

Figure 12. Configuring additional parameters (**Network** tab)

Adding a passthrough PCI device to a VM requires a full memory reservation. Unfortunately, the template's VM memory reservation is not retained during cloning within a vRealize Automation environment. To resolve the issue, add the customized property **VMware.Memory.Reservation** with a fixed-memory value. This is done using the **Properties** tab as shown in Figure 13. To add the property, click **new**, specify the property name (VMware.Memory.Reservation) with a value equal to the memory size provided on the **Machine Resources** tab. Set **encrypted** and **show in request** to "no" and **overridable** to "yes."

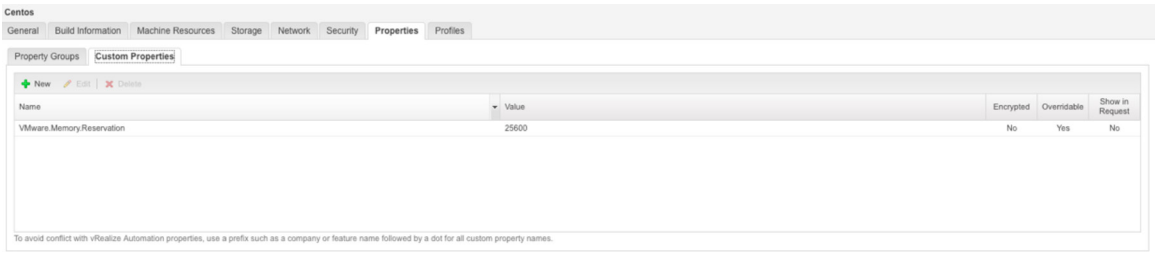

Figure 13. Configuring additional properties (Properties tab)

As a final step to create the blueprint definition, add the TensorFlow software component to the blueprint by selecting **software components** from the **categories** section and the dragging the TensorFlow software component onto the CentOS machine on the design canvas. Click **finish**. See Figure 14.

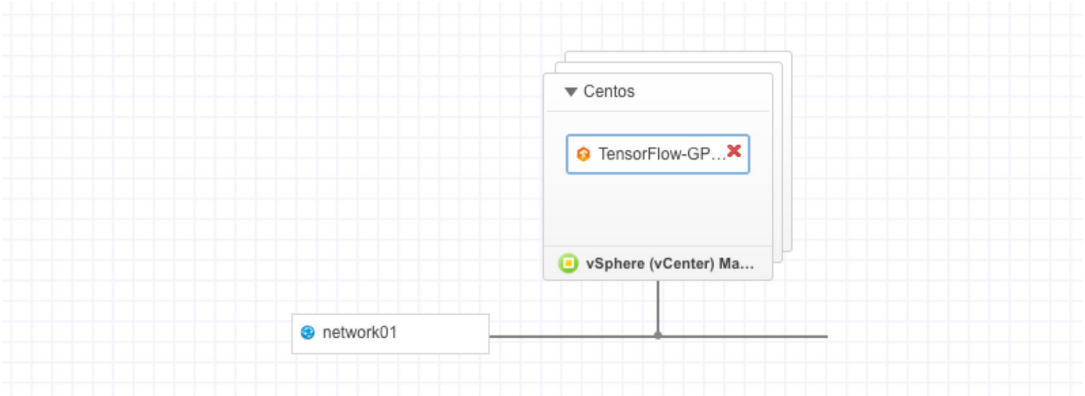

Figure 14. Adding TensorFlow to the blueprint

3) Publish blueprint
With the blueprint now defined, we can publish it as service in the vRealize Automation service catalog under the machine learning category and entitle users to access it.

1. In **Blueprints** select **Tensorflow-GPU**
2. Click **Publish**

Figure 15 shows several available ML/DL services with and without GPUs from a sample portal that data scientists can use to self-provision customized resources.

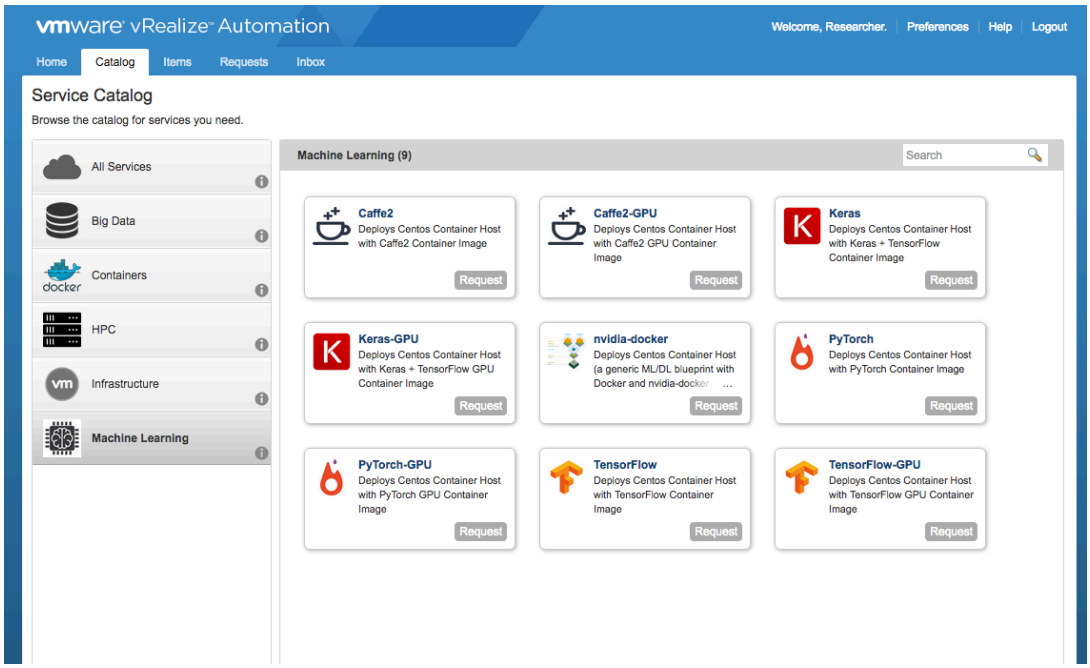

Figure 15. Machine Learning services in a data scientist portal

4.4 Sizing Considerations

Due to current vRealize Automation restrictions, VMs must be sized carefully to avoid over-provisioning of resources when using the blueprint to create vGPU-enabled VMs. Specifically, a set of identical and identically-configured hosts should be used to create a homogeneous pool of GPGPU hosts. In addition, the number of CPUs configured in the TensorFlow blueprint should be chosen to force vRealize Automation to place the correct number of vGPU-enabled VMs on each host. This VM size will be determined by the vGPU profile that will be used on the GPU cluster. For example, a P100 supports four 4Q-profile GPUs. As a result, the VM CPU count should be set in a manner that allows four VMs to fit on a GPU host without over-provisioning. Similarly, a P100 supports eight 2Q profiles and, in this case, the blueprint VM should be set such that each VM is given 1/8 of the host's CPUs.

4.5 Performance Recommendations

The following are general guidelines to achieve optimal virtual performance for computationally intensive workloads.

4.5.1 Host BIOS

In order to allow ESXi the most flexibility in using power-management features, set the power policy at the BIOS level to **OS controlled**.

Node memory interleaving should be disabled in order to allow the hypervisor to detect NUMA and apply NUMA optimizations.

While hyper-threading is often disabled when running HPC workloads, it should be enabled when running such workloads on vSphere. However, it is important to size VMs based on the number of physical cores, ignoring hyper-threads. More to the point, the sizing considerations described earlier should only take physical cores into account. Thus, when creating four VMs on a 16-core host (32 hyperthreads), each VM would be given four vCPUs, as opposed to eight.

4.5.2 Hypervisor

There are four ESXi power management policies: **high performance, balanced** (default), **low power** and **custom**. Though **high performance** power management would slightly increase performance of latency-sensitive workloads, in situations in which a system's load is low enough to allow Turbo to operate, it will prevent the system from going into C/C1E states, leading to lower Turbo Boost benefits. The **balanced** power policy will reduce host power consumption while having little or no impact on performance. It's recommended to use this default setting.

Virtual NUMA (vNUMA) exposes NUMA topology to the guest OS, allowing NUMA-aware OSs and applications to make efficient use of the underlying hardware. This is an out-of-the-box feature in vSphere 6.5. Prior to 6.5, use **VM Settings -> VM Hardware -> CPU -> Cores Per Socket** to force the virtual topology to be identical to the bare-metal topology. Set this value equal to the number of physical cores (not hyper-threads) per CPU socket, or to the desired size of the VM, whichever is smaller.

5. CONCLUSION

ML/DL workloads are becoming increasingly important to many organizations and the use of high-end GPU cards or other PCI devices to accelerate these workloads is essential to delivering timely results. These workloads can be self-provisioned by end users in a VMware virtualized environment using private-cloud or hybrid-cloud solutions. In this white paper, we describe two ways of utilizing GPGPU in the vSphere environment: DirectPath I/O and vGPU, and provide detailed instructions for preparing ML-as-a-service on a vRealize Automation private cloud with NVIDIA GRID vGPU for compute acceleration.

Appendix - Hardware and Software Details

HARDWARE	
Platform	Supermicro SYS-1028GR-TRT
Processor	Dual 12-core Intel Xeon E5-2600 v3 processors@2.5GHz (Haswell)
Memory	128GB DDR4
GPGPU	NVIDIA Tesla P100
SOFTWARE	
VMWare	
ESXi hypervisor	6.5u1
vCenter management server	6.5
vRealize Automation	7.3
OS	
OS Distribution	Centos 7.2
Kernel	3.10.0-327.el7.x86_64
NVIDIA GRID DRIVER	
ESXi Driver	NVIDIA-VMware-384.73-1OEM.650.0.0.4598673.x86_64.vib
Linux Driver	NVIDIA-Linux-x86_64-384.73-grid.run
Docker	17.09.1.ce
nvidia-docker	2.0

Contributors

Na Zhang is member of technical staff working on HPC within VMware's Office of the CTO. She has been working on various vHPC topics, including proof-of-concepts, performance benchmarking and tuning, design of vHPC tools and integration of HPC middleware with VMware products. Na received her Ph.D. degree in Applied Mathematics from Stony Brook University in 2015. Her research primarily focused on design and analysis of parallel algorithms for large- and multi-scale simulations running on supercomputers.

Shawn Kelly is a Staff Engineer focused on HPC and Machine Learning. He works directly with customers to develop innovative solutions in research computing. Shawn received Master of Science degrees in both Software Engineering and Information Technology Management from the Naval Postgraduate School in 2010, with a focus on cloud computing, and subsequently served at Marine Corps Systems Command developing solutions for the Marine Corps.

Josh Simons is the Chief Technologist for HPC, with more than 20 years of experience in the field. He currently leads an effort within VMware's Office of the CTO to bring the value of virtualization to HPC. Previously, he was a Distinguished Engineer at Sun Microsystems with broad responsibilities for HPC direction and strategy. He joined Sun in 1996 from Thinking Machines Corporation, a pioneering company in the area of Massively Parallel Processors (MPPs), where he held a variety of technical positions. Josh has worked on developer tools for distributed parallel computing, including language and compiler design, scalable parallel debugger design and development, and MPI. He has also worked in the areas of 3D graphics, image processing, and real-time device control. Josh has an undergraduate degree in Engineering from Harvard College and a Masters in Computer Science from Harvard University. He has served as a member of the OpenMP ARB Board of Directors since 2002.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com